웹시스템설계 실습문서 Lab 07

최지헌 unidev@ajou.ac.kr
이재현 dlwogus8888@ajou.ac.kr

2024-11-05

목차

1 학습 목표

- React 애플리케이션에서 상태 관리를 위해 state 개념과 useState function을 이해한다.
- Node.js 백엔드에서 반환한 값을 React 기반의 프론트엔드 애플리케이션에서 동적으로 렌더링하는 방법을 이해한다.

2 React state

React에서 말하는 state(이하 상태)란 리액트 컴포넌트가 소유하고 있는 데이터를 말한다. 어느 컴포넌트의 상태는 그 컴포넌트만이 접근하고 관리할 수 있어야 한다. 객체 지향 프로그래밍에서 각 객체가 자기만의 필드 값들을 가지는 것과 이 데이터는 그 객체만이 접근하고 수정할 수 있어야 한다는 점과 일맥상통한다.

2.1 Define state

2.1.1 함수형 컴포넌트의 상태 정의

함수형 컴포넌트를 선호한다면, hook을 사용하여 정의할 수 있다.

- Hook은 리액트 버전 16.8부터 추가된 요소로 클래스형이 아닌 함수형 컴포넌트에서도 여러 react의 기능을 사용할 수 있게 해 주는 요소이다.
- React 16.8.0은 Hook을 지원하는 첫 번째 배포이다. 업그레이드 시 React DOM을 포함한 모든 패키지의 업데이트를 진행해야 한다. React Native는 v0.59부터 Hook을 지원한다.

함수형 컴포넌트에서 상태를 정의하기 위해서는 hook 중 useState hook을 사용한다.

```
import React, { useState } from 'react';
function Example() {
    // "count"라는 새 상태 변수를 선언합니다.
    const [count, setCount] = useState(0);

    return (<p></p>)
}
```

useState 함수는 인자로 표현할 상태값의 초기값을 받고, 반환값으로 현재의 상태값과 그 상태 값을 업데이트 할 수 있는 상태 갱신 함수를 쌍으로 제공한다. const [a, b] = [1, 2]와 같은 형태의 문법은 구조 분해 할당(destructuring)으로 불리며 아래에서 더 많은 정보를 찾아 볼 수 있다. https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

useState 함수는 상태값 한 개를 정의하므로 여러개를 정의하고 싶다면 여러번 호출하면 된다.

```
function ExampleWithManyStates() {
    // 상태 변수를 여러 개 선언했습니다!
    const [age, setAge] = useState(42);
    const [fruit, setFruit] = useState('banana');
    const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
```

```
    // ...
}
```

2.2 상태값 접근

2.2.1 함수형 컴포넌트의 상태값 접근

함수형 컴포넌트에서는 useState 메서드가 반환한 배열의 첫 번째 인덱스에 들어있는 값으로 참조하면 된다. 다시 사용할 수 있도록 이 값을 함수 내의 다른 변수에 할당해 놓고 사용한다.

```
function Example() {
    const [count, setCount] = useState(0);

    return (
        <div>
        <p>You clicked {count} times</p> <!-- 위에서 count 변수에 받았기에 count라는 이름으로
            참조합니다. -->
        </div>
    );
}
```

2.3 상태값 수정

함수형 컴포넌트를 사용할 때 제공된 메서드를 쓰지 않고 상태값을 직접 갱신해서는 안 된다.

2.3.1 함수형 컴포넌트의 상태값 수정

함수형 컴포넌트를 사용한다면 useState를 통해 상태를 정의할 때 반환된 배열의 두 번째 엘리먼트에 담겨 있는 함수로 상태를 갱신할 수 있다.

```
import React, { useState } from 'react';

function Example() {
    const [count, setCount] = useState(0);

    return (
        <div>
        <p>You clicked {count} times</p>
        <button onClick={() => setCount(count + 1)}> //setCount로 count의 기존값에 1을 추가한다.
            Click me
        </button>
        </div>
    );
}
```

상태 변경 함수가 자꾸 2번씩 호출되는 상황 관련 React Strict 모드가 활성화 되어 있는 경우 상태 변경 처리는 2번 반복해서 처리된다다. React는 상태를 immutable로 취급하기에 개발자가 상태 변경 시 상태를 immutable로 취급하게끔 강제하기 위해 의도적으로 상태 갱신을 2번 반복한다. 적절하게 코딩하였다면 2번 반복된 상태 갱신 처리에서도 정확한 값으로 갱신되어야 한다.

상태 변경 함수가 2번 호출되는 것 때문에 상태값의 증감이 2번씩 처리되어 의도한 값이 나오지 않는다면 Strict 모드를 비활성화 하거나 상태 갱신을 immutable하게 처리하여야한다.

2.4 상태 변경과 렌더링

리액트는 상태가 변경될 경우 자동으로 해당 컴포넌트를 다시 렌더링하여 DOM에 변경된 내용이 반영될 수 있도록 한다. 여러번의 상태 업데이트가 있을 경우 매번 렌더링하지 않고 이를 묶어 한 번의 상태 갱신으로 처리한 다음 렌더링을 하여 렌더링에 의한 성능 저하를 최소화하려고 시도한다. 상태를 변경할 때 상태 변수에 직접 접근하여 값을 업데이트 하는 것이 아니라 상태 변경 함수를 이용하는 이유가 이것이다.

2.5 연속적인 상태 변경 이슈

위에서 설명되었듯이 상태 변경 함수는 실행 즉시 상태를 업데이트 하지 않는다. 즉 상태 변경은 비동기적으로 작동한다. 따라서 아래와 같은 코드는 의도대로 작동되지 않을 가능성이 있다.

```
//처음에 this.state.counter = 0이라고 가정

this.setState({
    counter: this.state.counter + 1
});

this.setState({
    counter: this.state.counter + 1 // 이 시점에서 this.state.counter는 아직 0이다!
});

this.setState({
    counter: this.state.counter + 1 // 이 시점에서 this.state.counter는 아직 이다!
});
```

프로그래머는 위 함수 실행 후 최종적으로 this.state.counter의 값이 3이 되기를 기대했겠지만, this.state.counter의 값은 3이 아닌 1로 업데이트 된다. 이러한 비동기적 상태 업데이트 이슈는 연속적인 상태 변경 외에도, 상태를 업데이트 하는 데에 상태값이 필요한 경우에도 발생할 수 있는 이슈이다.

리액트에서는 이전 상태에 의존하는 상태 갱신을 지원하기 위해 setState 메서드에 함수가 주어질 경우 첫 번째 인자로 현재까지 갱신된 상태값을 넣어 호출하는 방식을 지원한다. 이 함수의 반환값은 새로 생신된 상태값이어야 한다.

```
//처음에 this.state.counter = 0이라고 가정

this.setState(function(state) {
    return {
        counter: state.counter + 1
```

```
    };
});

this.setState(function(state) {
    return {
        counter: state.counter + 1 //이 시점에서 state.counter는 1이다.
    };
});

this.setState(function(state) {
    return {
        counter: state.counter + 1 //이 시점에서 state.counter는 1이다.
    };
});
```

최종적으로 this.state.counter는 3으로 업데이트 된다. 함수형 컴포넌트를 사용하여 useState를 통해 상태 갱신 함수를 받았을 경우에도 마찬가지로 상태 갱신 함수에 값이 하닌 이전 상태값을 파라미터로 받고 반환값으로 새 갱신값을 주는 함수를 주면 된다.

```
const [value, setValue] = useState(0);
setValue(function(prev){
    return prev + 1;
});
```

3 React Event

3.1 함수형 컴포넌트의 이벤트 핸들러 정의

```
function App() {
    function sayHello() {
        alert('Hello!');
    }

    return (
        <button onClick={sayHello}>
        Click me!
        </button>
    );
}
```

3.2 이벤트 핸들러 연결

기본 HTML 엘리먼트의 이벤트 프로퍼티에 함수 레퍼런스를 주는 것으로 연결할 수 있다. 필요한 것은 함수 레퍼런스이므로 화살표 함수로 기술하는 일도 가능하다.

```
function App() {
    function sayHello() {
        alert('Hello!');
    }

    return (
        <button onClick={sayHello}>
        Click me!
        </button>
    );
}
```

```
function App() {
    return (
        <button onClick={()=>{alert("Hello!")}>
        Click me!
        </button>
    );
}
```

3.2.1 기본 동작 방지하기

HTML의 input 엘리먼트 몇몇은 이벤트가 발생되었을 때 브라우저가 처리하는 기본 동작을 가지고 있는 경우가 있다. 상황에 따라 리액트 쪽에서 이 기본 동작을 무효시키고 자신의 처리를 진행해야 할 필요가 있다. 이 경우 이벤트 헨들러 함수의 첫 번째 파라미터로 넘어오는 이벤트 객체의 preventDefault 메서드를 호출한다.

```
function Form() {
    function handleSubmit(e) {
      e.preventDefault();
      console.log('You clicked submit.');
    }

    return (
      <form onSubmit={handleSubmit}>
        <button type="submit">Submit</button>
      </form>
    );
}
```

3.2.2 인자 전달하기

루프 내부의 엘리먼트에서는 같은 동작을 파라미터에 따라 약간 다르게 처리해야 할 필요가 있다. 이럴 경우 이벤트 핸들러에 추가적인 매개변수를 전달하여 처리한다.

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>
```

```
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

두 가지 형태 모두 가능하며, deleteRow의 첫 번째 파라미터에는 id값이 오게 된다.

3.2.3 하위 컴포넌트의 이벤트 처리하기

react에서 이벤트를 던지는 것은 기본 HTML 엘리먼트 뿐이다. 아래는 그저 하위 컴포넌트에게 onClick이라는 이름의 props로 onButtonClick이라는 함수의 레퍼런스를 전달할 뿐이다.

```
function Example(){
    function onButtonClicked() {
        console.log("Clicked!");
    }

    return (
        <CustomElement onClick={onButtonClicked}></CustomElement>
    )
}
```

하지만, 함수가 props를 통해 전달 가능하므로 하위 컴포넌트에서 이벤트가 발생했을 시 이를 잡아 props를 통해 전달받은 이벤트를 통해 함수를 대신 호출해 주는 형태는 가능하다.

```
function Example(){
    function onButtonClicked(){
        console.log("Clicked!");
    }

    return (
        <CustomElement onClick={onButtonClicked}></CustomElement>
    )
}

function CustomElement(props){
    return (
        <button onClick={props.onClick}>클릭</button>
    )
}
```

4 실습 과제

4.1 프로젝트 디렉토리 구조

```
Lab07_(본인학번)/
├── frontend
│   ├── (생략)
│   ├── src
│   │   ├── (생략)
│   │   ├── App.css
│   │   ├── App.js
├── backend
│   ├── main.js
│   ├── (생략)
```

※ frontend 디렉토리는 npx create-react-app frontend 후 생성된 디렉토리이다.
※ bb에 제공된 App.css의 내용을 기존 ./frontend/src/App.css에 덮어쓴다.
※ ./frontend/src/App.css와 ./backend/main.js는 각각 스타일링과 server 관련 파일로, 수정하지 않는다.
※ node_modules 디렉토리를 삭제한 뒤 압축하여 제출한다.

4.2 프로그램 구현 내용

react를 활용하여 수강 신청 프로그램의 client 단을 구현한다.

4.3 사전 정보

다음 파일들은 사전에 제공된 것으로, 수정하지 않는다.

- ./backend/main.js
  - courses
    - 강의 목록을 저장하는 배열이다.
    - 강의 정보는 순서대로 id, 강의명, 학점, 현재 신청한 학생 수, 수강 정원을 의미한다.
  - `[GET]` /api/courses
    - 강의 목록 정보를 반환하는 API이다.
  - `[PUT]` /api/courses/:id/enroll
    - 강의 신청 요청을 처리하는 API이다.
    - 해당 강의의 현재 신청한 학생 수를 1 증가시킨다. 단, 정원 초과 시 현재 신청한 학생 수를 변경하지 않고 에러 응답을 반환한다.
    - 존재하지 않는 id 조회 상황은 고려하지 않는다.
  - `[PUT]` /api/courses/:id/cancel
    - 강의 취소 요청을 처리하는 API이다.

    * 해당 강의의 현재 신청한 학생 수를 1 감소시킨다.
    * 존재하지 않는 id 조회 상황은 고려하지 않는다.
  - Port 8000에서 listen 한다.

- ./frontend/src/App.css
  - 웹 페이지 스타일링을 정의한 css 파일이다.
  - 스타일링이 모두 적용된 화면은 Figure 1과 같다.

Figure 1: 초기 화면

4.3.1 구현 요구사항

./frontend/src/App.js

- server의 /api/courses 엔드포인트에 GET 요청을 보내 강의 목록(courses)를 받아와 화면에 표시한다.
  - 강의 목록에서 웹 시스템 설계(4학점 / 40명 / 40명)은 각각 강의 명, 학점, 현재 신청한 인원 수, 최대 신청 가능 학생 수(수강 정원)을 의미한다.
- 신청 버튼을 누를 시 /api/courses/:id/enroll 엔드포인트에 PUT 요청을 보내 강의 정보를 변경하고, 즉시 화면에 다음 변경 사항들이 반영돼야한다.
  - 버튼의 글자가 "신청"에서 "취소"로 변경된다.
  - 강의 목록 밑의 신청한 강의명과 학점 정보가 추가된다.
  - 신청한 강의의 학점만큼 총 학점이 증가된다.
  - 단, 정원 초과 시 위의 변경 사항들을 반영하는 대신 정원 초과를 알리는 alert 창을 띄워야한다.
- 취소 버튼을 누를 시 /api/courses/:id/cancel 엔드포인트에 PUT 요청을 보내 강의 정보를 변경하고, 즉시 화면에 다음 변경 사항들이 반영돼야한다.

  - 버튼의 글자가 "취소"에서 "신청"으로 변경된다.
  - 강의 목록 밑의 취소한 강의명과 학점 정보가 제거된다.
  - 취소한 강의의 학점만큼 총 학점이 감소된다.

- App.css에 정의된 스타일링을 적용해야한다. (단, 정의된 10개를 모두 적용하지는 않아도 되나, 이들 중 최소 5개를 적용해야한다.)
- 상세 동작 화면은 첨부 파일의 lab07 영상을 참고한다.

4.4 Submission

제출 기한: 11월 05일 자정 / 지각 제출: 11월 07일 자정

5 Appendix

5.1 프론트엔드와 백엔드 연동

- 프론트엔드와 백엔드 domain 혹은 port가 다른 경우, API를 요청 시 CORS(Cross-Origin Resource Sharing) 문제가 발생한다.
- 이를 해결하기 위한 방법으로는 여러 가지가 있지만, 이번 실습에서는 http-proxy-middleware를 사용한다.

5.2 API 연결로 인한 CORS 문제 해결

src/setupProxy.js

```
const { createProxyMiddleware } = require('http-proxy-middleware');

module.exports = function (app) {
    app.use(
        createProxyMiddleware('/api', {
            target: 'http://localhost:8000',
            changeOrigin: true,
        }),
    );
};
```

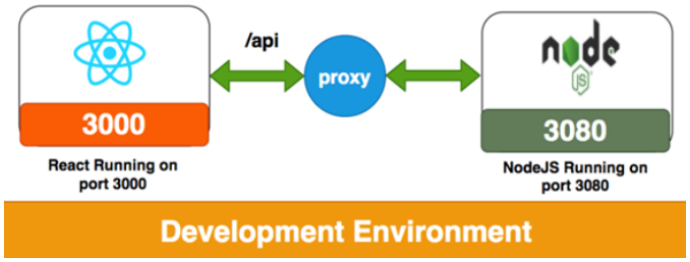

Figure 2: proxy 설정

5.3 Fetch data with React useEffect

HackerNews 사이트 검색 API를 이용하여 데이터를 가져오는 예제이다. useEffect hook 함수에서 fetch를 이용해 백엔드 서버로부터 값을 가져온 뒤에 렌더링한다.

```
import React, { useState } from 'react';

function App() {
    const [data, setData] = useState({ hits: [] });

    useEffect(async () => {
        fetch('http://hn.algolia.com/api/v1/search?query=redux', {
            method: 'GET'
        }).then(response => response.json())
        .then(result => {
            setData(result.data);
        })
    }, []);

    return (
        <ul>
        {data.hits.map(item => (
            <li key={item.objectID}>
            <a href={item.url}>{item.title}</a>
            </li>
        ))}
        </ul>
    );
}

export default App;
```

본 실습에서는 처음 렌더링 시 강의 목록 데이터를 받아와 화면에 표시할 수 있도록 아래의 코드를 그대로 사용한다.

```
useEffect(() => {
    fetch('/api/courses')
      .then(response => response.json())
      .then(data => setCourses(data))
      .catch(error => console.error('Error fetching courses:', error));
  }, []);
```